django-teamwork Documentation

Release 0.0.1

Les Orchard

Jun 11, 2017

Contents

django-teamwork is a Django app that offers an authorization backend with support for per-object permissions based on combinations of Teams, Roles, and Policies.

- Source code on GitHub
- Build status on travis-ci ()
- Latest documentation on Read The Docs (source)

Contents:

CHAPTER 1

Overview

What is django-teamwork?

django-teamwork is a Django app that offers an authorization backend with support for per-object permissions based on combinations of Teams, Roles, and Policies.

This app was born out of Kuma, the Django-based wiki / CMS that powers the Mozilla Developer Network. MDN hosts a large body of documentation, with pages arranged into a tree of sections and sub-sections. These site sections are each managed by different teams and people, whom in turn have their own requirements for controlling access to read and alter content.

So, django-teamwork was created to provide per-object and per-section flexibility for controlling permissions granted by content objects. Here are some highlights:

- Teams can be given ownership of content Objects
- Teams offer Roles that, when assigned to Users, grant selected Privileges for team-owned content objects
- Independent of Teams and Roles, Policies can be set on content objects that grant Permissions based on criteria such as:
  - whether a User is anonymous or authenticated;
  - whether the User owns the object;
  - or by matching specific Users and Groups.
- Policies can be set on a Site objects to apply site-wide
- Policies can be specified in `settings.py` to establish a base set of Permissions for the entire installation.
- Content objects can optionally filter the set of Permissions granted by Teams, Roles, and Policies to add or remove Permissions based on custom model logic.
- Content objects with a hierarchical tree structure can optionally offer a list of parents. This is used to implement inheritance for Team ownership and Policy application, so that Permissions cascade down the content tree.

CHAPTER 2

Getting Started

Installation

First, get the package itself installed. You may find it handy to try this:

```
pip install -e 'git://github.com/lmorchard/django-teamwork.git#egg=django-teamwork`
```

This may or may not work, depending on whether I've yet done my job in building a sensible `setup.py`.

(Pull requests welcome! See also: *Contributing*)

Configuration

Add `teamwork` to your `INSTALLED_APPS` list in `settings.py`:

```
INSTALLED_APPS = (
    # ...
    'django.contrib.auth',
    'teamwork',
    # ...
)
```

Add `teamwork.backends.TeamworkBackend` to `AUTHENTICATION_BACKENDS` in `settings.py`:

```
AUTHENTICATION_BACKENDS = (
    # ...
    'django.contrib.auth.backends.ModelBackend',
    'teamwork.backends.TeamworkBackend',
    # ...
)
```

Finally, create all the models:

```
$ ./manage.py syncdb
$ ./manage.py migrate teamwork
```

Of course, your mileage may vary, if you’re not using South to manage your model changes.

CHAPTER 3

Usage

Contents under construction; this page is mainly a feature teaser.

The best sample code is currently found in:

- the teamwork_example app used in tests;
- and in the backend tests themselves.

Checking permissions for a user and content object

@@ TODO. It goes a little something like this:

```
if not request.user.has_perm('wiki.view_document', doc):
    raise PermissionDenied
```

Using the `get_object_or_404_or_403` shortcut

@@ TODO. It goes a little something like this:

```
from teamwork.shortcuts import get_object_or_404_or_403
# ...
doc = get_object_or_404_or_403('wiki.add_revision', request.user,
    Document, locale=document_locale, slug=document_slug)
```

Base policy in `settings.py`

@@ TODO. Example. Here's an inadequate sample:

```
TEAMWORK_BASE_POLICIES = {
    'anonymous': (
        'wiki.view_document',),
    'authenticated': (
        'wiki.view_document', 'wiki.add_document', 'wiki.add_revision'),
}
```

Setting a Policy on a Site

@@ TODO. Example test.

Setting a Policy on a content object

@@ TODO. Example test.

Creating Teams and Roles

@@ TODO. Example test.

Granting Team ownership of content Objects

@@ TODO. Example model and test.

Assigning Roles to Users

@@ TODO. Example test.

Filtering permissions with per-object logic

@@ TODO. Example model and test.

Supporting content hierarchies and Permission inheritance

@@ TODO. Example test and model.

CHAPTER 4

Contributing

More to come here, soon. See also, *TODO*.

Pull requests welcome!

Hacking notes

- Setting up a virtualenv:

```
virtualenv ./test-venv
. ./test-venv/bin/activate
pip install -r requirements-test.txt Django
```

- Running tests:

```
./teamwork_example/manage.py test teamwork
```

- To continually check pep8, tests, and coverage while working on OS X:

```
gem install kicker
kicker -c -e ./run-tests.sh teamwork teamwork_example
```

- Running the example site:

```
./teamwork_example/manage.py syncdb --noinput; ./teamwork_example/manage.py␣
↪runserver
```

- To regenerate `test_data.json` from example site:

```
./teamwork_example/manage.py dumpdata -n --indent=4 sites auth.user teamwork wiki␣
↪> teamwork_example/fixtures/test_data.json
```

CHAPTER 5

TODO

- Move all the below into GitHub Issues, once this gets to feature-complete
- Get this onto PyPi and make sure the usual methods of installation work
- Add views for Team and Policy management, outside of Admin
  - Need views in example app for profile views
- Popup-friendly views?
  - to apply / adjust Policy on a content Object
  - to assign a user to one of your Teams
- API ergonomics
  - shortcut to convert from codename + content object to Permission
- Support ForeignKey for Policy directly from content objects?
- Support many-to-many for Policies and content objects?
- Abstract out / make more flexible some of the integration points
  - optional fields & methods on content objects
    - team field
    - get_permission_parents, get_all_permissions
- Consider optimizations for mass-lookup cases, because this does nothing for that now.

Use Cases / Specs

This is a thinking-aloud section where I braindumped about what I'm trying to accomplish here:

- As a creator of a content Object I want to create a Team In order to delegate Permissions granted by a content Object

- As a creator of a content Object I want to assign ownership of my Object to a Team In order to share ownership of a content Object
- As a manager of a Team I want to create a Team Role that encompasses a subset of my Permissions In order to delegate some, but not all, Permissions granted by an Object
- As a manager of a Team I want to assign a Role on my Team to another User In order to delegate Permissions granted by Team-owned Objects
- As the manager of a Role, I want to be given a list of my Permissions that are available to delegate, So that I can easily build a Role
  - How to assemble this list? Can't be as permissive as superuser access, can only consist of Permissions available to Team creator
- As a manager of a content Object, I want to be able to create a Policy that encompasses a set of Permissions, In order to delegate Permissions to Users who are not Team members
- As a creator of content Objects in a hierarchical tree, I want Team ownership to apply recursively down through the tree, In order to avoid assigning Team ownership to each child Object individually
- As a creator of content Objects in a hierarchical tree, I want a Policy to apply recursively down through the tree, In order to avoid assigning a Policy to each child Object individually

CHAPTER 6

Indices and tables

- genindex
- search